White Paper Report

Report ID: 98517

Application Number: HD5082209

Project Director: Andrew Ashton (Andrew_Ashton@brown.edu)

Institution: Brown University

Reporting Period: 1/1/2010-6/30/2012

Report Due: 9/30/2012

Date Submitted: 3/14/2012

Semantically Rich Tools for Text Exploration: TEI and SEASR

White paper
HD-500822-09
Project Director: Andrew Ashton
Brown University

March 14, 2012

The ability to reuse tools across projects is an increasingly important feature of many digital humanities efforts. Researchers using the Text Encoding Initiative (TEI) guidelines have traditionally published to the web using processes such as XSLT transformations, which are intended to reproduce a work as an enhanced digital surrogate using HTML, PDF, or another publication format. [1] Processes such as these often contain the complete pipeline of logic that produces a publishable product, whether the product is a document, a data set, or visualizations. Other approaches to scholarly software design offer opportunities to enhance the reusability of scholarly software tools by breaking apart these processing pipelines into smaller, more general-purpose chunks and providing an environment to blend and align tools from different domains. To test this approach, the Brown University Library's Center for Digital Scholarship (CDS) and the Brown University Women Writers Project (WWP) have developed a prototype suite of software tools to explore TEI encoded texts in the Software Environment for the Advancement of Scholarly Research (SEASR).[2] This whitepaper describes the rationale for creating TEI tools for SEASR, the challenges encountered during the process, and information about the software produced by this project.

SEASR is a software environment for creating and sharing text and data mining tools. Textual analyses and visualizations created in SEASR can be shared on the web and adapted by other scholars to support their own research. The characteristic of SEASR that permits such exploration and interchange is its modular approach to software architecture. Software modularity encourages sharing and reuse of discrete pieces of software within a broader framework. It enables relatively simple software tools to be combined to produce results that were previously the domain of complex and sometimes proprietary software. We sought to offer the same benefits to the TEI community by incorporating TEI-specific, semantically focused tools into the SEASR ecosystem.

Within the SEASR environment, analysis is modeled as a "flow": a pipeline, or a series of "components" arranged to produce a specific analysis (see figure 1). Each component in a flow is a small piece of software that ingests and processes data, and then passes it to the next component in the flow. Typically, each component performs a single function, such as loading data, tokenizing words, or stemming. But, through their recombination and re-sequencing

---

[1] Text Encoding Initiative (http://www.tei-c.org)

[2] Brown University Library Center for Digital Scholarship (http://library.brown.edu/cds);
Brown University Women Writers Project (http://www.wwp.brown.edu);
The Software Environment for the Advancement of Scholarly Research (SEASR, http://www.seasr.org), funded by the Andrew W. Mellon Foundation, is a modular, open-source software framework for text analysis and visualization.

SEASR components enable innumerable potential analyses. In fact, it is assumed that SEASR flows will draw on components that originated in other projects, and which may have been designed to deal with different types of data, but which, because they were designed as standalone modules, ultimately serve many analytical purposes. In order to facilitate this kind of sharing and exploration, SEASR components are meant to be stored in open repositories, either in the SEASR project repository at NCSA, in a public code repository, and/or in locally hosted instances of the software. Locally installed SEASR servers can access repositories remotely, and automatically ingest the available components to use locally.

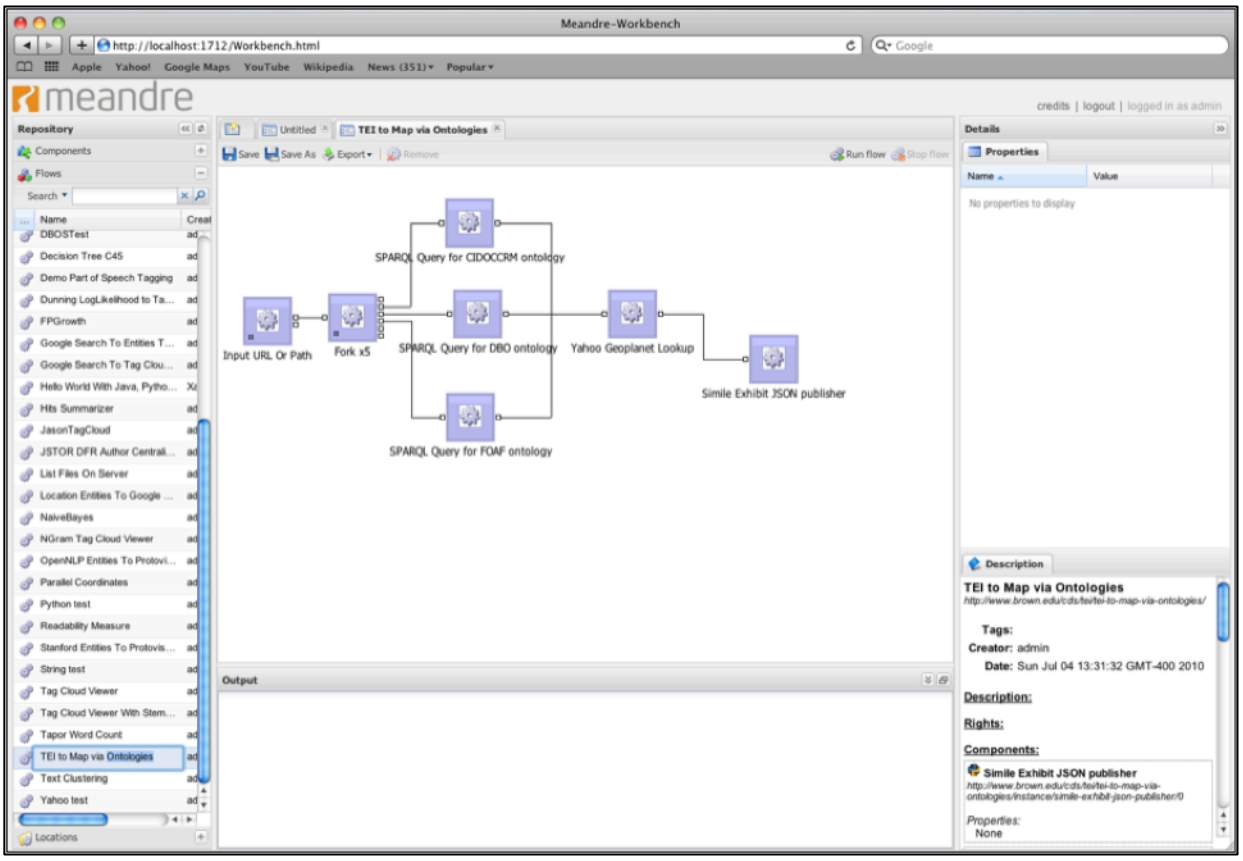


Figure 1. Designing a SEASR "flow"

Once a flow of components is constructed, its results can be made publicly available on a web site, in a publication, and sometimes made available to rerun using different data. The SEASR framework offers an execution engine, which permits flows to be run via a web application, such as a publication interface of a digital project (as a way of working with project data). More commonly, flows are used as part of an independent web-based workbench, included with the SEASR software, through which a scholar could examine data drawn from multiple projects. And because flows can be flexibly combined and altered, they may permit much greater latitude in scholarly inquiry, instead of channeling text analysis into predetermined sequences. For example, using the tools developed as part of this project (Appendix A), a literary scholar interested in examining how a particular editor modified the stage directions in a play could construct a flow to extract and visualize stage directions from the Diplomatic rendition of a TEI document and then easily rerun the same analysis using the Editor X filter (see Figure 2).

Pipeline processes for transforming TEI documents have been common for a long time. XProc is a pipeline language that is, as of May 2011, a W3C Recommendation.[3] Similarly, the Orbeon XForms project developed the XML Pipeline Language (XPL) to provide similar functionality in their software.[4] For web publishing, Apache's Cocoon has been a mainstay of the XML community for years.[5] SEASR's approach to pipeline processing is different in that it is not essentially an XML pipeline, but a data pipeline more generally. SEASR's components can, in principle, perform many of the functions that languages like XProc or XPL can perform, but there is no starting assumption that the purpose of a SEASR flow is simply to transform XML. This is an important distinction, because it allows XML data to be manipulated similarly to other data formats. A component developed for the purpose of tokenizing a string, for example, has as much potential utility in a flow intended to transform a TEI text as it does in a flow designed to analyze raw OCR data. SEASR adds to the pipeline model the potential for a community of shared components, developed to serve a particular need, but generalized enough to be applicable to data from different sources. Because SEASR's architecture supports and encourages the sharing of tools, it provided a suitable platform for a project focused on developing modular tools for the institutionally and geographically diverse community of TEI users.

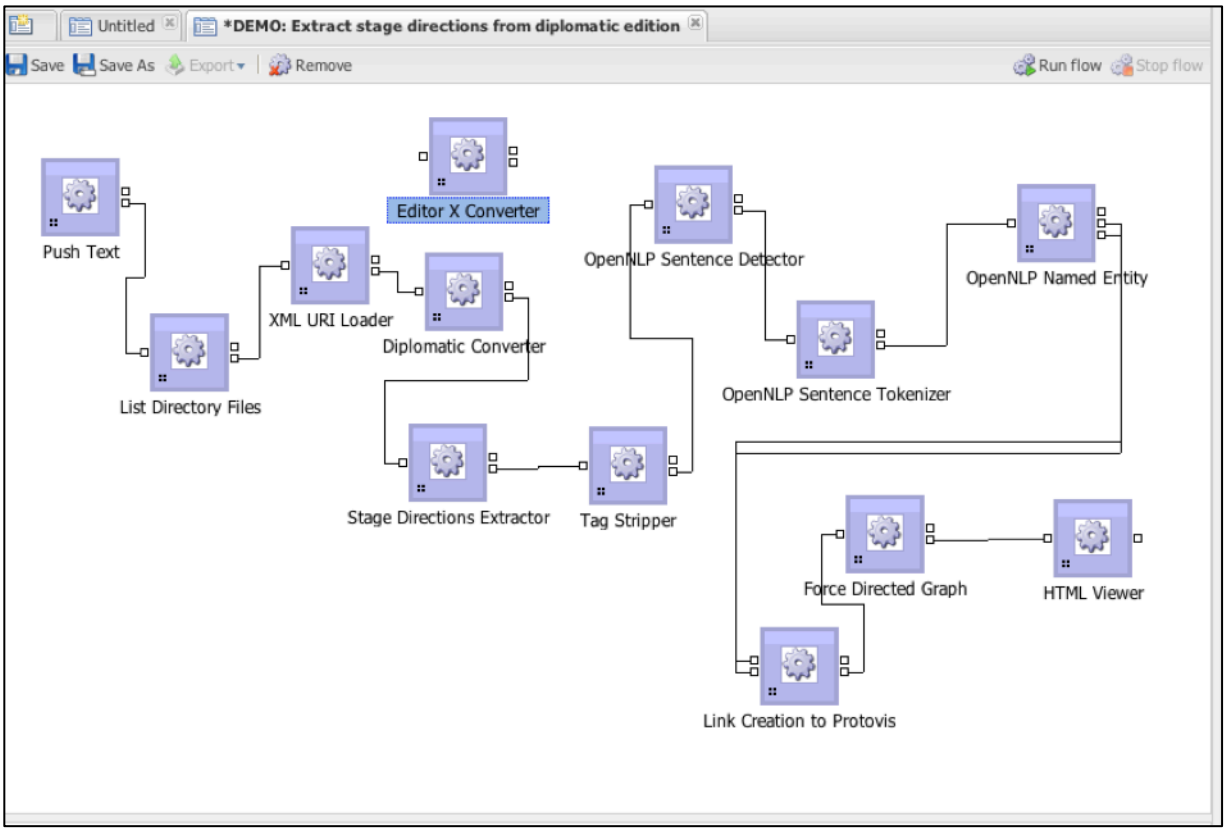


Figure 2. Creating visualizations of data within diplomatic and edited versions of a text

[3] http://www.w3.org/TR/xproc/

[4] http:/wiki.orbeon.com/forms/doc/developer-guide/xml-pipeline-language-xpl

[5] http://cocoon.apache.org/

Working with TEI in SEASR

This project started with the idea that TEI's capacity for storing rich semantic data made it a good candidate for the kinds of analyses that SEASR flows afford. TEI's markup logic makes the semantic information encoded in a text processable for the purposes of transformation, extraction, analysis, and reuse. When used in tandem with other data mining and visualization tools available in the SEASR environment (e.g., Simile, OpenNLP, ProtoVis), these TEI components for SEASR can offer a new way to explore questions germane to literary and textual scholars. We set out to create a suite of tools that approached the notion of data mining and visualization from the perspective of small, crafted corpora rather than large data sets. We sought to determine whether components within a framework like SEASR could be designed and reused easily enough to allow the framework to be of equal value for exploring questions about semantic data in small corpora as it is for crunching text in large data sets.

"Data mining" is a term traditionally associated with the extraction and manipulation of data from large corpora, often with little or no markup. Tools like SEASR, designed for working with "big data," have proliferated with the intention of providing researchers with flexible and scalable software for processing large data sets. Carefully crafted TEI collections have, thus far, been at the periphery of these developments. Because it requires the rationalization of text data and some human intervention, TEI data is commonly created on a smaller scale than the data produced via mass digitization efforts such as Google Books or Hathi Trust. TEI has the benefit of being able to provide detailed modeling of text data, and therefore offers analytical possibilities that are distinct from those afforded by big sets of undifferentiated data. Google Books data, for example, is massive, and information derived from it may be used to support broader hypotheses created during the study of smaller corpora.[6] Users of "big data" may seek to identify characteristics of corpora at a macro-level in the hopes of finding correlations to hypothesized historical, sociopolitical, or genre-specific trends. TEI data, on the other hand, is most frequently scoped to the context of the document or the curated corpus, and the data contained within the TEI corpus describes in greater detail the semantic and structural information within the text, as well as a record of editions, corrections, commentary, and other paratexts. Much of the work that had gone into SEASR prior to this project was well-suited to the examination of big data, while the tools to examine small corpora in ways that might yield questions to apply to bigger data sets had not yet been create.

SEASR components are designed to handle data serially. Each component expects data to be input in certain formats and will output in certain formats, and the connectivity of individual components depends on their ability to take in and put out like data formats (e.g., strings, Java maps arrays, JSON, XML, etc.) Because of this design, SEASR is hypothetically able to deal equally well with data derived from large data sets as it is from smaller, more crafted data sets. While the Meandre architecture (the underlying engine that runs the SEASR software) is designed to scale to support intensive analysis in high performance computing environments, the model of moving data serially from an input source through to a flow result is as relevant in both cases.

---

[6] Michel, J., Shen, Y., et al. "Quantitative Analysis of Culture Using Millions of Digitized Books." Science 14 January 2011: Vol. 331 no. 6014 pp. 176-182. DOI: 10.1126/science.1199644. http://www.sciencemag.org/content/331/6014/176

Approaches to developing general-purpose TEI software

Among the chief challenges to creating modular and shareable software for TEI is TEI's inherent flexibility. TEI is not a standard intended to facilitate data interoperability as much as it is a standard for creating, as Syd Bauman puts it, "expressive" and machine-operable data designed to operate in specific contexts.[7] As a result, TEI data is often not suited to reuse without some sort of intervention, analysis, and transformation. Because this project sought to produce tools that would be reusable across many TEI projects and schemas, this factor presented a daunting challenge. How would we provide tools that are of use in a variety of TEI projects, that allow a deep investigation of semantic data in the texts, and that require minimal intervention by the user?

Several other projects have grappled with the same issue - notably, the MONK project, which produced the TEI-Analytics format as a solution to this problem.[8] TEI-Analytics is a variant of TEI that seeks to facilitate the sharing and aggregation of data from disparate corpora by identifying common elements to allow for analytical software (such as MONK) to process. TEI-Analytics allowed the MONK project to normalize their texts prior to downstream processes such as tokenization and morphosyntactic analysis. A key difference between our work and that of the MONK project is that MONK intended to combine a known set of texts for use with the MONK tools, while our SEASR components are intended for use in other projects, with little customization or intervention. Because we did not want to impose the requirement of using a standard format such as TEI-A on the potential users of our components, MONK's approach was inapplicable as a general solution to this issue.

Another approach developed by the MONK project uses a technique called "schema harvesting" to examine DTDs or schema from TEI collections in order to determine what needs to happen to the text in order to be usable by the MONK software. The MONK team developed a tool called "Abbot" for this purpose, and documented the tools on their website.[9] While the Abbot software itself is designed to facilitate the conversion of TEI to TEI-A for the purposes of feeding into MONK, their approach was intriguing. The notion of schema harvesting for the purpose of learning about the capabilities of a text with respect to an analysis framework is very appealing. Indeed, our initial specification for the TEI components for SEASR included a key component that would, in effect, take the same approach as Abbot and provide some feedback to the user about which components were applicable to their texts. Ultimately, we took a different approach and our version of a schema harvester component was never created. The main reason is that the idea of vetting texts with reference to a predetermined set of valid features is at odds with SEASR's fundamental premise. How could we know what kind of components a user might have available for their analysis? Would the introduction of such an organizing principle to the TEI components for SEASR lead to their functioning as a silo of tools within the broader community, and would this short-circuit the possibilities for mashing-up tools and data afforded by SEASR?

---

[7] Bauman, Syd. "Interchange vs. Interoperability." *Basilage: The Markup Conference 2011: Proceedings.* August 2-5, 2011. http://www.balisage.net/Proceedings/vol7/html/Bauman01/BalisageVol7-Bauman01.html

[8] Pytlik-Zillig, Brian. "TEI Analytics: converting documents into a TEI format for cross-collection text analysis." Literary and Linguistic Computing 2009 24(2):187-192; doi:10.1093/llc/fqp005

[9] http://monkproject.org/docs/abbot.html

These considerations led to a conclusion that the best approach for a framework such as SEASR was not to restrict incoming TEI data. Instead, we explored a new approach to standardizing operable data outside of the TEI guidelines; an exploration which yielded useful insights about the intersection between TEI and the Semantic Web, but which ultimately proved overly complex to implement in this project. We presented at Digital Humanities 2010 on our idea to use a "semantic map" to define what semantic data the SEASR components could process, and to allow users to map data from their own encoding schemes to the vocabularies specified in the semantic map.[10] This approach establishes non-TEI ontologies as translators between TEI projects with varying schemes for encoding semantic data and the SEASR components developed as part of this project.

The semantic map concept was modeled on the approaches of previous projects working with semantically rich TEI collections, most notably the work of the Henry III Fine Rolls Project, based at King's College London. The creators of that project developed a method for using RDF/OWL to model the complex interpersonal and spatial relationships represented in their TEI documents.[11] In the Henry III Fine Rolls Project, TEI data is mapped to external ontologies, including CIDOC-CRM and SKOS.[12] This approach serves well as a model for mapping some of the more nebulous concepts of interest to the SEASR components to the variable encoding of those concepts in TEI.

In order to demonstrate how a semantic map might be used, we created a prototype application to transform a TEI document into any number of RDF serializations of specific subsets of semantic data (e.g., place data, personal names, etc.) The prototype used a Django web application to render TEI documents as XHTML with RDFa (see figure 3).[13] The RDF can then be distilled out of the XHTML to produce serializations of data defined according to any ontology. Using this sort of web application, any web-accessible TEI document can have any number of RDF serializations that may be addressed via a URL and used by external programs such as SEASR. The source code for this prototype application is available in the project's GitHub repository.

While the semantic map offers a great deal of flexibility for matching locally-defined data structures with general-purpose software tools, the task of creating such an abstract layer of functionality proved to be beyond the scope of this project. Our examination of this approach made clear a number of problems. First, as we became more familiar with the SEASR development environment, it became clear that there was no obvious way to package the prototype semantic mapping application as a set of SEASR components. Our application was written in Django, a framework that uses the Python programming language. Although SEASR ostensibly supports writing components in Python, we discovered that SEASR's engine,

---

[10] Ashton, Andrew. "Semantic Cartography: Using RDF/OWL to Build Adaptable Tools for Text Exploration." Presented at Digital Humanities 2010 (London, UK, July 2010). http://dh2010.cch.kcl.ac.uk/academic-programme/abstracts/papers/html/ab-760.html

[11] Vieira, Jose Miguel and Ciula, Arianna. "Implementing an RDF/OWL Ontology on Henry the III Fine Rolls." Presented at OWLED 2007 (Innsbruck, Austria, June 2007). http://www.webont.org/owled/2007/PapersPDF/submission_6.pdf

[12] CIDOC-CRM:http://www.cidoc-crm.org/. SKOS: http://www.w3.org/2009/08/skos-reference/skos.html

[13] RDFa is an XHTML format which embeds semantic web data in attributes within the HTML. The specification is available here: http://www.w3.org/TR/rdfa-syntax/

Meandre, uses a version of Python running under Java, known as Jython. Early in our development, Meandre only supported an outdated version of Jython, which lacked many essential standard libraries that were used in the semantic map prototype. Developers with the SEASR project helpfully offered to create a custom version of Meandre that used a newer version of Jython, but that solution may have prevented other instances of SEASR from using the tools that resulted from this project. The other solution was to build out the prototype web application to server as a non-SEASR service for mapping TEI data to RDF. However, SEASR is already a new and relatively complex tool-set, and the prospect of deploying a second web application in tandem with SEASR would add an additional dimension of abstraction and indirection, and would be unrealistic for many potential users.

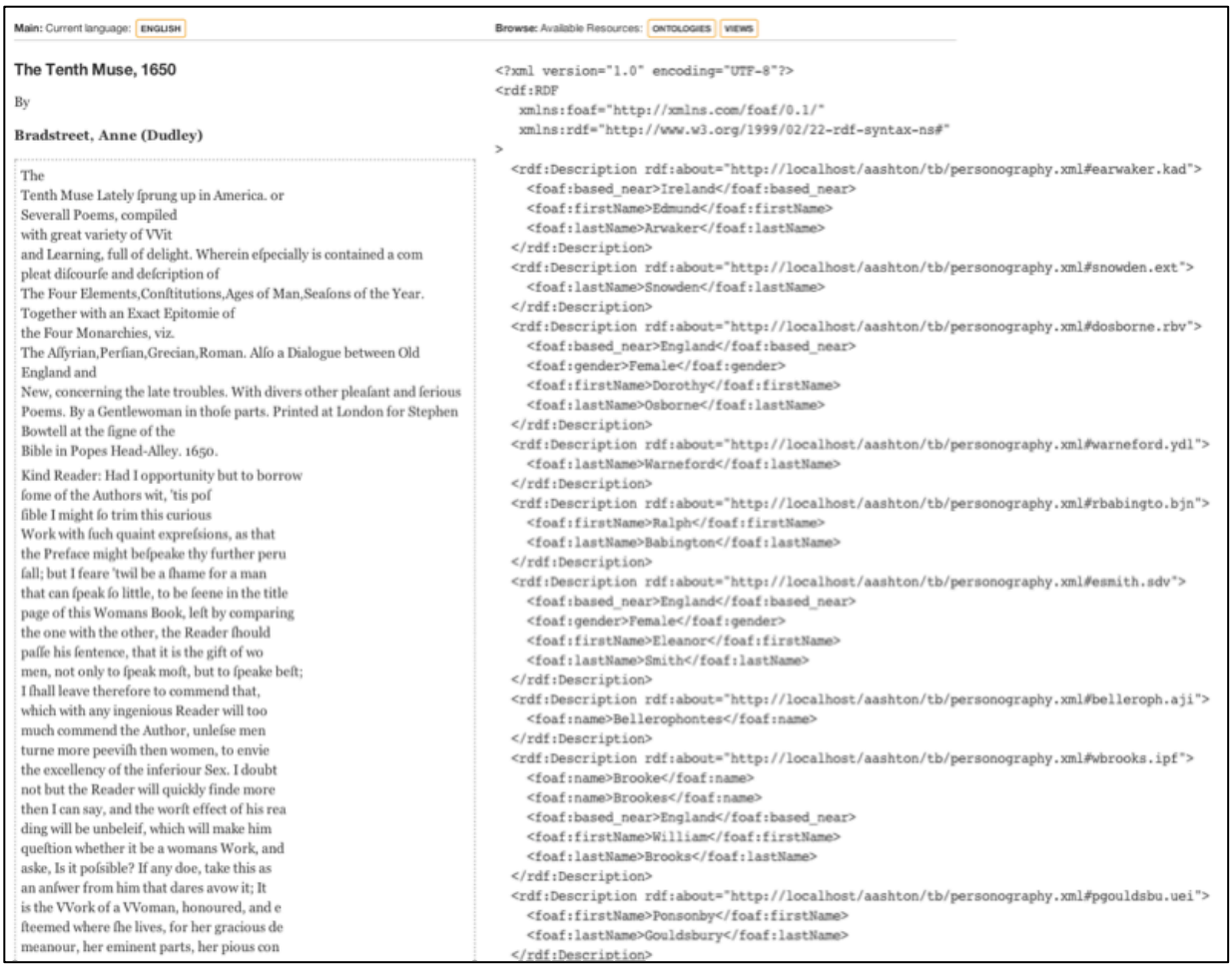


Figure 3. Mapping names to the FOAF ontology.

Second, the prototype semantic mapping application required that users provide their own XSL stylesheets to transform their data so that it could be converted to RDF by the application. While the expertise to create XSL may be available to many TEI projects, and the application could be optimized to accept a wide variety of transformed documents, this requirement highlighted the key shortcoming of the semantic map approach: that the onus of defining the expression of semantic information within the TEI would still fall to the individual project or user. The approach of using a semantic map, while elegant from a software-development

perspective, introduced an unnecessary barrier to entry for users. With this lesson in hand, we shifted our approach and opted to develop components that inherently offer users the flexibility to derive relevant data from their TEI whether their local encoding practices mirror the expected TEI standards or not.

After examining two promising approaches - schema harvesting and semantic mapping - we determined that the most sensible and most easily implemented solution for our purposes was also the simplest: we would follow the SEASR model and break our components down to a level of granularity where users with different TEI encodings could easily rebuild our exemplars to work with their own data. This meant first developing a set of components intended to work with XML at a very basic level. These components, described in Appendix A under "Foundational components", offer the basic building blocks for manipulating XML. They include basic functions such as retrieving XML via the web or a file system, combining groups of XML documents into a single document for processing purposes, XSLT transformation, and validation against RelaxNG schemas. They also include filtering components, which allow for documents that do not meet given criteria (such as the those that do not contain data at a specified XPath or do not validate) to be filtered out of a collection being analyzed by a flow. These basic components complement many XML components that already existed as part of the SEASR project's component repository.[14]

With these foundational components in place, we set out to build another group of components intended to operate on commonly used features of TEI in order to provide for a more deliberate selection of data from encoded literary texts. These components, described in Appendix A under "Document filters," parse specific features of the TEI that have been determined to be indicative of different views or renditions of the selected document. For example, the "Diplomatic" document filter component parses an incoming TEI document and passes through the document as it appears on the page without revision or editorial intervention of any kind. In order to do this, the component uses some of the aforementioned test and filtering component code to:

- Suppress the <note> element by anyone except the author.
- Use the content of <sic>, <barb>, <orig> elements within <choice> elements.
- Choose the highest-certainty value when <unclear> or <seg> appear in <choice>.
- Suppress <supplied>.
- Includes content of all revision element.

The output of this component will be a pared down XML document that now represents the diplomatic view of the text in question, and to which any further operations and analyses will be applied. Some of the document filters can be used serially with each other, to apply finer filtering to the document. For example, one could conceivably filter a document using the "Authorial Text" filter (to remove any apparatus or notes by other contributors) and then use the "Modernized" filter to use only the modernized spellings of words within the resulting document. Other combinations would result in contradictory filtering. If one were to put the "Normalized" filter before the "Diplomatic" filter in a flow, any changes made by the "Normalized" filter would remain, even if they were contraindicated by the "Diplomatic" filter.

[14] http://repository.seasr.org

A third group of components provides some ways of further extracting and transforming data for use with other analysis and visualization components. These components work with specific semantic data within the TEI, extracting and passing along specific genres within a literary text, (such as poems of different types), certain snippets of text (such as spoken passages), and discrete data (such as place names). They also provide some tools for converting the data to prepare for use with other components available in the SEASR ecosystem. One such component geocodes place names using a Google API, while another converts data to JSON for publication in a Simile Exhibit. Other components available to SEASR users can be attached in order to convert XML or strings of resulting text to data formats that can be used for any number of other graphs, visualizations, or other analytical outcomes (see Appendix B for a few sample flows using these components alongside other available SEASR components). Like the document filters, these data extraction and analysis filters have certain criteria hard-coded into them; criteria that are based on observations of known textbases and with reference to the TEI guidelines.

The latter two groups of components - document filters and extraction and analysis components - provide a convenient way for scholars to work with texts using common approaches to TEI encoding. The decision to craft this particular set of filters using these criteria was based on a number of factors. We considered known sets of highly detailed TEI data, such as the Women Writers Project textbase, in addition to sample texts provided from other projects at the University of Nebraska, and a broad examination of TEI data freely available online. The specification for each of these filters was based on practices recommended in the TEI guidelines. However, we recognize that there are endless variations both on the kinds of filters a researcher might want to apply to a text, and on the customizations a textbase might implement and still be working within the TEI guidelines. Therefore, many of the components from these two groups can be roughly approximated and altered using the foundational components from the first of the three groups. For example, the diplomatic filter essentially implements a set of predetermined choices about the features of the TEI that represent the diplomatic view of the text. In applying this filter to a textbase whose encoding practices would lead to a different set of choices for displaying the diplomatic view, a user can use some of the foundational components, such as Choice Handler and XPath Find and Replace, to alter or override the results of the stock diplomatic component (see figure 4). Because many of the filtering components contain more advanced processing, such as assigning variables based on text data and then filtering using those values, which is difficult to handle using XPath, it is recommended to place the stock filtering components before any additional XPath filters. The filtering components will simply pass through the data unchanged if they do not match any of the criteria for filtering. This approach offers the greatest flexibility to accommodate some less common uses of the TEI and local customizations, while still offering the convenience of some stock components to work within the standard P5 specifications.

Challenges

Working with TEI in the SEASR environment presented a number of challenges, some of which are detailed in the above descriptions of approaches to creating general-purpose TEI software. The most obvious example of a potential problem is inconsistent or erroneous markup. Collections with these issues will encounter problems using many kinds of analytical software, and it is beyond the scope of this project to solve that problem. Even with more carefully

encoded collections, there are still many uses of TEI to which these components will not easily apply. The creation of a set of abstract, highly granular XML components (i.e., the "foundational" components) addresses some of the native flexibility of the TEI standard, but this project focused exclusively on the encoding of literary texts. The foundational components can be applied to any TEI, but the components have not been tested on non-literary collections.

Dealing with out-of-line or referential TEI data structures, such as personography or feature structures, presents another significant challenge. TEI allows many types of data to be stored in standalone lists, in external documents, or in the TEI header and referenced via ID attributes or other keys. Processing these data proved to be difficult for a number of reasons; they include the variety of standalone structures that may be used within the guidelines, the possibility to store data both inline as out of line, the possibility of customization within referenced structures, and the option to store a referenced data structures in included files (or even as includes of includes, etc.). These and other considerations contribute to a range of possibilities that are too numerous to fully account for in the scope of this project. In these situations, it is recommended that the TEI be transformed using the XSLT component to pull data from referenced data structures and put them in-line with the rest of the document text for processing by SEASR. Future development efforts focused on better support for out-of-line data would greatly enhance the usefulness and convenience of these tools.

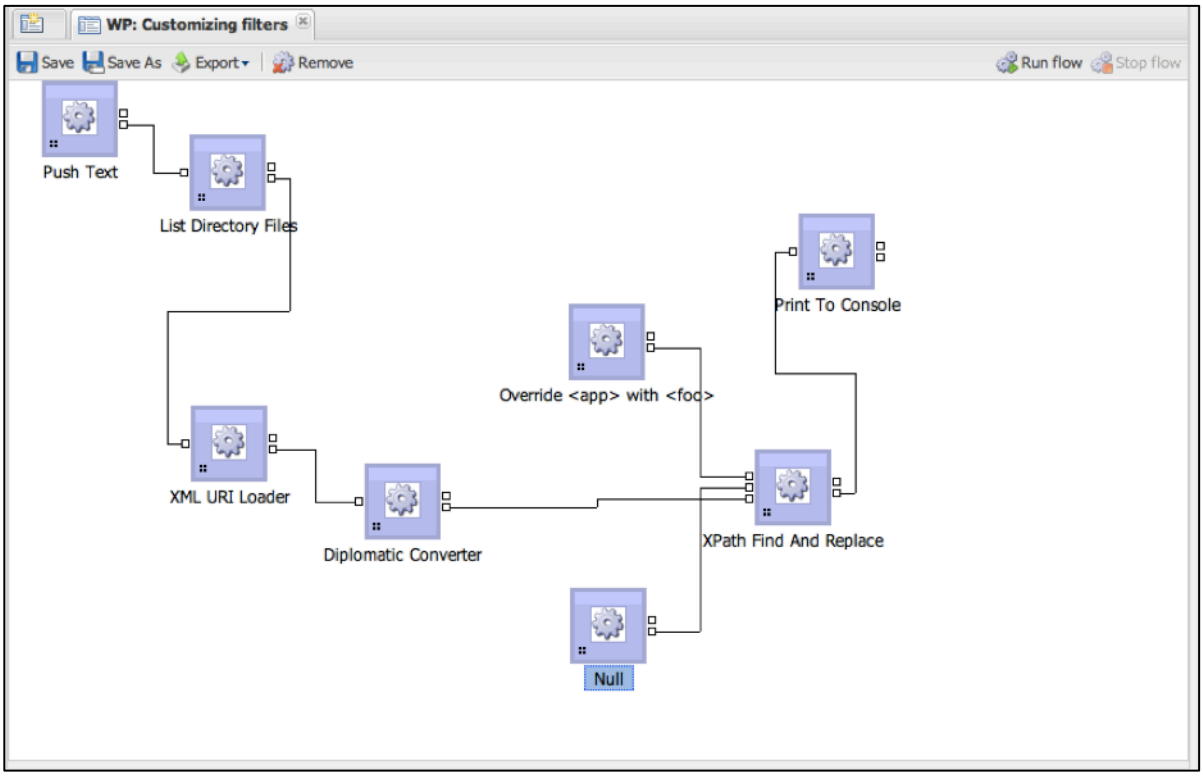


Figure 4. Customizing document filters.

Finally, the Meandre software itself is still under development, and this was the cause for several bottlenecks in the development process. Some issues with the Meandre software have already been described (i.e., the lack of support for recent versions of Jython). Furthermore, it was difficult to understand some of the basic approaches used by the architecture in its handling

of data. For example, it was not immediately clear how a flow handles the processing of many incoming documents, considering that a flow is naturally a serialized process. Do the documents get processed one-by-one and the results queued up prior to display? Is it up to the creator of the flow to create loops to process every document? Do the components we create have to handle the iterations themselves? It turned out that Meandre uses an internal process that determines when the work of one component has completed, and passes the results to the next component, although it took some trial and error, and eventually the help of the SEAR team to fully understand the underlying rules. There are several other internal rulesets (firing policies for components, for example) that are not immediately clear to external developers. The SEASR team was immensely helpful with all of these issues, and they would likely benefit from more feedback from external developers to help identify which information needs to be made more explicit in the documentation.

In a similar vein, we have encountered a number of performance and stability issues while using the Meandre software. We encountered frequent problems with memory leaks, processes not stopping or restarting, and severe slowness in some seemingly simple processes. These challenges could have been due to component design, network interruptions, or any number of factors. Their presence is not indicative of a flaw in Meandre, but rather highlights the complexity of working within a software environment, particularly one in which much of the software is created by a variety of independent developers. The SEASR team is regularly releasing new versions, which frequently fix some of the issues we encountered. However, there have been issues with sets of components and dependencies becoming incompatible across versions of Meandre. Components that work in Meandre 1.4.8 might not work in 1.4.9 without reviewing and revising each of the dependencies to ensure compatibility. Occasionally, the best approach to encountering components that stopped working was to delete and rebuild the instance of Meandre from scratch - a time consuming process. These issues may make it problematic to use the Meandre workbench with novice or even intermediate-level users.

Conclusion

The many TEI projects that exist in the literary research community would benefit immensely from the development of common toolsets aimed at unlocking the semantic richness of these encoded collections. The great strengths of the TEI - its scope and flexibility - are the same things that make it difficult to create general-purpose TEI software. Frameworks such as SEASR offer excellent opportunities to pair TEI data that has been extracted from its original context with general data analysis tools, thus circumventing the need to design sophisticated software that only works with TEI. To this end, this project produced an initial suite of SEASR tools to facilitate the further integration of TEI into the SEASR environment.

This process did not unfold without some difficulties, most of which are addressed within this paper. Ultimately, several of the more difficult questions faced during this work – particularly those surrounding the reuse and generalizability of TEI data - are of immediate interest to the TEI research community. Debates about interchange, interoperability, and reuse have infused the community of TEI users for a long time, and although this project did not identify a single solution to these persistent problems, it did produce a valuable examination of some possible approaches, as described previously. This examination was especially useful in the context of an already-established software platform such as SEASR. It allowed us to explore hypothetical approaches to handling diverse TEI data while grappling with the real constraints imposed by

the software. The semantic map is an example of the difficulty of translating a rapidly developed prototype into a scalable software environment. Despite the fact that such an approach proved infeasible, it has already generated interest from other projects, which might have more success based on the foundational laid here.

The final products of this project should have broad applicability to the TEI community, although some of the components in Appendix A are designed to work with known textbases, and serve as examples of how to construct components with the libraries provided. Furthermore, variations in applications of namespaces or TEI customizations may require more intervention or customization of these components by users. The approach to accommodating different encoding practices is intended to ease the entry into working with these tools. However, we acknowledge that the range of possibilities within the TEI could overwhelm the flexibility of these tools. Demonstrating such challenges with freely accessible software can have a beneficial side effect of adding to conversations within the TEI community to identify and implement conventions, including those such as TEI-A, to facilitate data reuse.

Finally, SEASR is new and complex software, and using it still requires a fairly high-level of technical expertise. This project struggled to identify with meaningful precision the audience that would use these components. Many scholars, even those with some digital training, may struggle with establishing and learning to use a framework like SEASR. Technologists who support scholars may have more luck in using the framework, provided they have an adequate understanding of the subject data in order to recognize which components and repositories would be applicable. Even with technical support in place, variations in versions of Meandre, local memory limitations, and the general complexity of the architecture mean that working successfully with SEASR is still a significant undertaking. Despite these challenges, the components that were produced through this work provide a foundation for the further development of modular data tools, either within or outside of the SEASR framework.

Appendix A: Software and descriptions

The software produced during this project is available on GitHub at https://github.com/Brown-University-Library/TEI-Components-for-SEASR/, and in the Brown Digital Repository at http://repository.library.brown.edu:8080/fedora/objects/bdr:11456/datastreams/ZIP/content. It is released under a GNU Public License.[15]

Sample flows are available at http://teicomponents.wordpress.com.

The SEASR components developed for this project fall roughly into three categories:

- Foundational Components: Components designed to read, merge, validate, and prepare document data for processing in a SEASR flow.
- Document filters: Pre-made components for filtering document data by aspect.
- Data extraction and analysis: Components for retrieving specific kinds of semantic data from processed texts.

[15] http://www.gnu.org/licenses/gpl.txt

Foundational components:

1. Choice Handler: Resolves <choice> elements in TEI according to the user's selection (i.e., choose "sic", "corr", "abbr", "expan", "orig", "reg", "am", or "ex").

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/choicecomponent

2. File-System Enumerator: Lists file URIs on a local file-system and allows for filtering of filenames using regular expressions. Used for choosing files to feed into a SEASR flow.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/filesystemenumeratorcomponent

3. Group Builder: Takes a series of documents and transforms them into a single instance of Java Strings.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/groupbuildercomponent

4. Line Splitter: Splits a string on a given delimiter and returns it as an instance of Java Strings.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/linesplittercomponent

5. RelaxNG Validator: Returns a Boolean value for each loaded document, reporting whether it validated against its Relax NG schema.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/relaxngcomponent

6. RelaxNG Filter: Filters the documents based on whether they validated against their Relax NG components.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/relaxngfiltercomponent

7. Tag Stripper: Strips tags out of XML and returns only the text nodes.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/tagstripper

8. TEI Corpus Bundler: Bundles a group of TEI documents into a single document.

   https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/teicorpuscomponent

9. XML Include Resolver: Sets the xml:base attribute for the root node of the document to the parent of the URI specified in the brown-seasr processing instruction from XMLURILoader and then processes the xml includes.

https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/xmlincluderesolvercomponent

10. XML URI Loader: Loads a series of XML documents from a list of URIs that may indicate locations on a filesystem or a HTTP service. Documents will be annotated with a processing instruction indicating where they were retrieved.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/xmluriloadercomponent

11. XPath Processor: Processes and incoming stream of XML documents against a given XPath string.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/xpathcomponent

12. XPath Filter: Filters an incoming stream of XML documents against a given XPath string.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/xpathfiltercomponent

13. XPath Find and Replace: Transforms incoming documents according to two parameters: an XPath expression selecting nodes to find and an XPath expression selecting nodes to replace them with from the context of the found node.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/xpathfindandreplacecomponent

14. XSLT 2.0 Processor: Uses Saxon Home Edition to do XSLT 2 transformations.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/xslt2component

Document Filters:

15. Authorially Revised: The document representing the author's final intentions-includes the content of <add> and suppresses the content of <del>, and of <subst>. Otherwise identical to the Diplomatic filter (see below).

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/authoriallyrevisedcomponent

16. Authorial Text: The text including only the words for which the author is directly responsible-no apparatus or notes. Suppress <fw> and any material identified with other contributors (e.g. printer, publisher, editor). Suppresses <add> or <del> by any hand other than the author.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/authorialtextcomponent

17. Author's First Intent: The document as originally written, without authorial revisions-includes the content of <del> and suppresses the content of <add>.

https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/authorsfirstintentcomponent

18. Diplomatic: The document as it appears on the page without revision or editorial intervention of any kind. Suppressed <note> by anyone except the author; within <choice>, uses content of <sic>, <abbr>, <orig> (suppress <corr>, <expan>, <reg>); where <unclear> or <seg> appear in <choice>, chooses the highest-certainty value; chooses children of <app> based on witness identification; suppressed <supplied>; includes content of all revision elements.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/diplomaticcomponent

19. Editor X: The document reflecting the editorial choices and input of a specific editor, as identified by an input value corresponding to the value for that editor in the TEI document(s). When processing <app>, uses readings selected by the chosen editor; includes <note> from the chosen editor.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/editorcomponent

20. Expanded: The document with all abbreviations expanded but in other respects identical to Diplomatic (see above).

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/expandedcomponent

21. Externally Revised: The text as revised by some external hand. Use <add> and <del> only when by the external hand.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/externallyrevisedcomponent

22. Modernized: The document with modernized spelling (provided through <reg>).

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/modernizecomponent

23. Normalized: The document as it appeared on the page, but edited by an editor. Within <choice>, uses content of <corr>, <expan>, <reg> and suppresses <sic>, <abbr>, <orig>. Chooses high-certainty <unclear> and <seg>. Includes content of all revision elements and content of <supplied>.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/normalizecomponent

24. Witness X: The document as represented in a specific witness. In <app>, use readings identified with a specific witness, as identified by an input value corresponding to the value for that witness in the TEI document(s).

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/witnesscomponent

Data Extraction and Analysis:

25. Dedication Extractor: Extract dedications for further processing: Any .

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/dedicationextractorcomponent

26. Geocoder: Takes place names as input and attempts to resolve and return lists of latitude and longitude pairs. Uses Google Maps API.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/geocodingcomponent

27. Letter/Notes to Reader Extractor: Extracts Notes to Reader: .

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/lettersextractorcomponent

28. Lines of Verse Extractor: Extract lines of verse of any type: Any <l> element.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/linesofverseextractorcomponent

29. Personal Names Extractor: Returns an XML document containing <persName> elements found in the text.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/personalnameextractorcomponent

30. Place Names Extractor: Returns an XML document containing <placeName> elements found in the text. Suppresses relative places (<offset>, <measure>).

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/placenameextractorcomponent

31. Poem Extractor: Extract verse included in a longer text. Selects any or <lg> element with one of the following @type attributes: poem, couplet, tercet, quatrain, quintain (poetry), sextet, septet, octave, nonet, decastich. Also selects any lg[@type="stanza"] if not already within one of the first two items and any milestone[@type="poem"] if not already within one of the first two items.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/poemextractorcomponent

32. Simile JSON Writer: Takes a stream of TEI XML documents and writes a JSON file for Simile to the filesystem containing geocode data.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/similejsoncomponent

33. Speech Extractor: Returns an XML document containing <q> or <sp> elements found in the text.

    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/speechextractorcomponent

34. Stage Directions Extractor: Extract stage directions/dramatic features using <stage>.

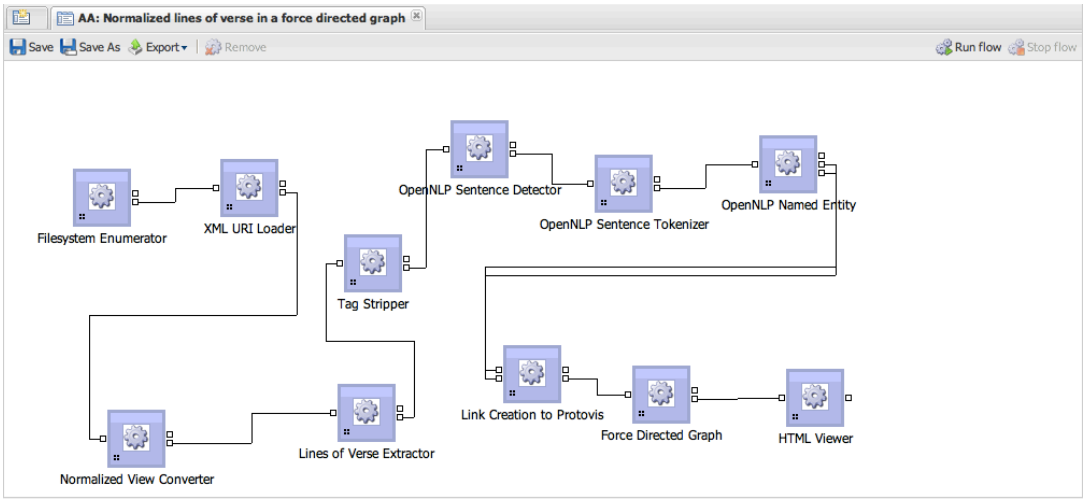
    https://github.com/Brown-University-Library/TEI-Components-for-SEASR/tree/master/trunk/src/main/java/edu/brown/seasr/stagedirectionsextractorcomponent

Appendix B: Sample flows

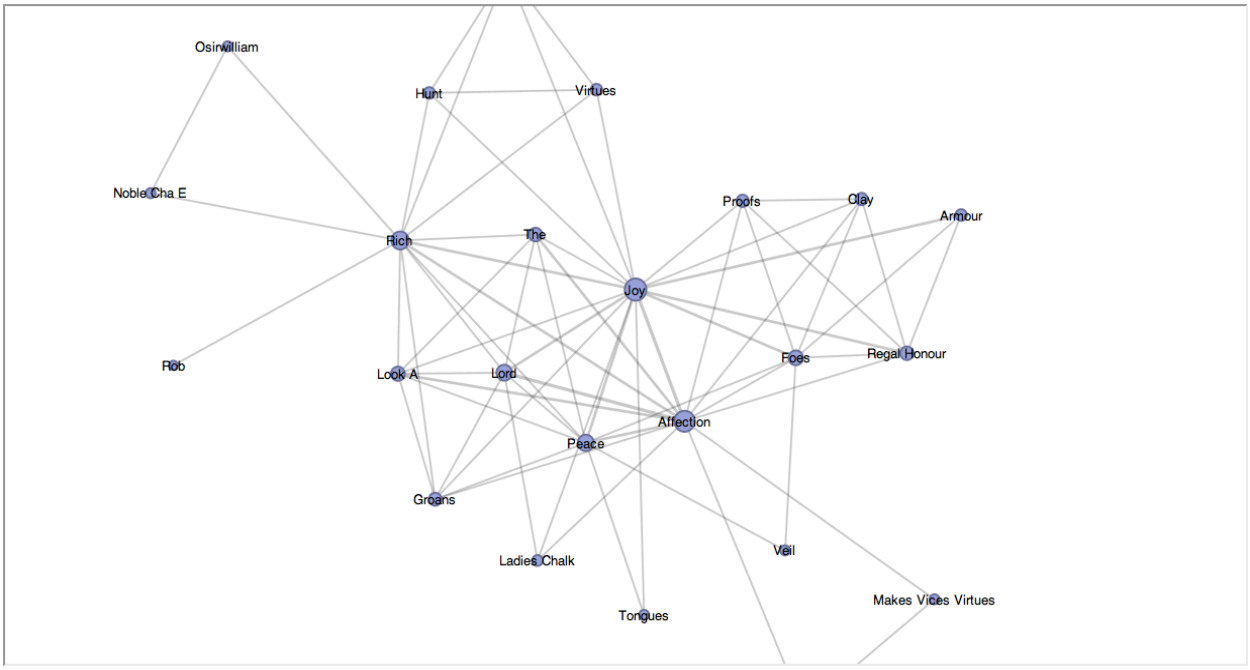
The following are a few sample flows that make use of both the components produced by this grant and components available in the general SEASR repository.

View words in lines of verse from normalized versions of a sampling of Women Writers Project textbase in a force-directed graph:

The following flow:

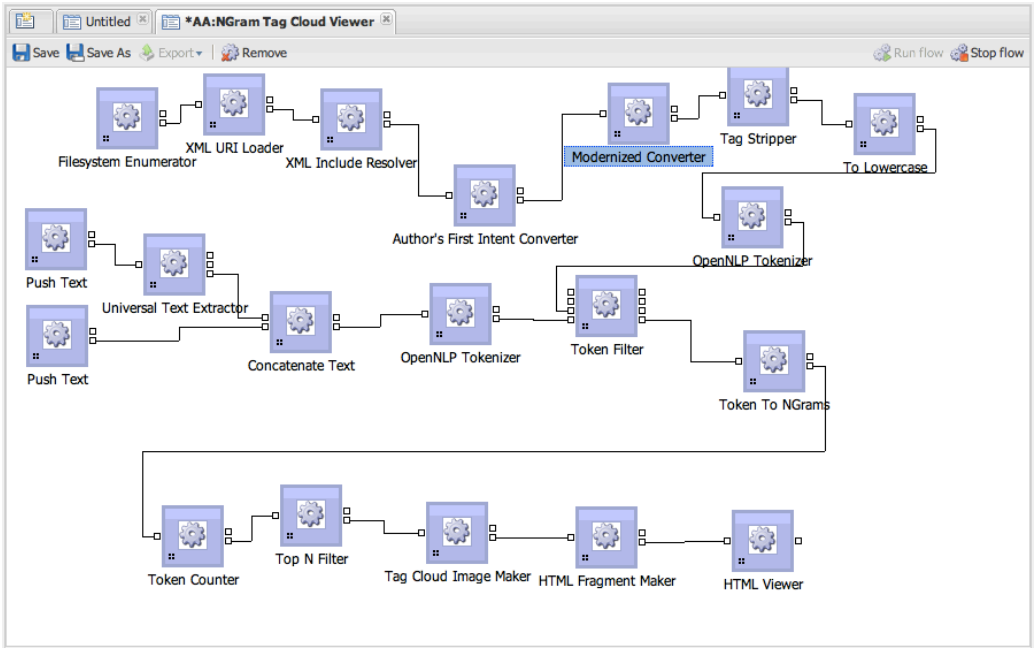


Produces the following result:

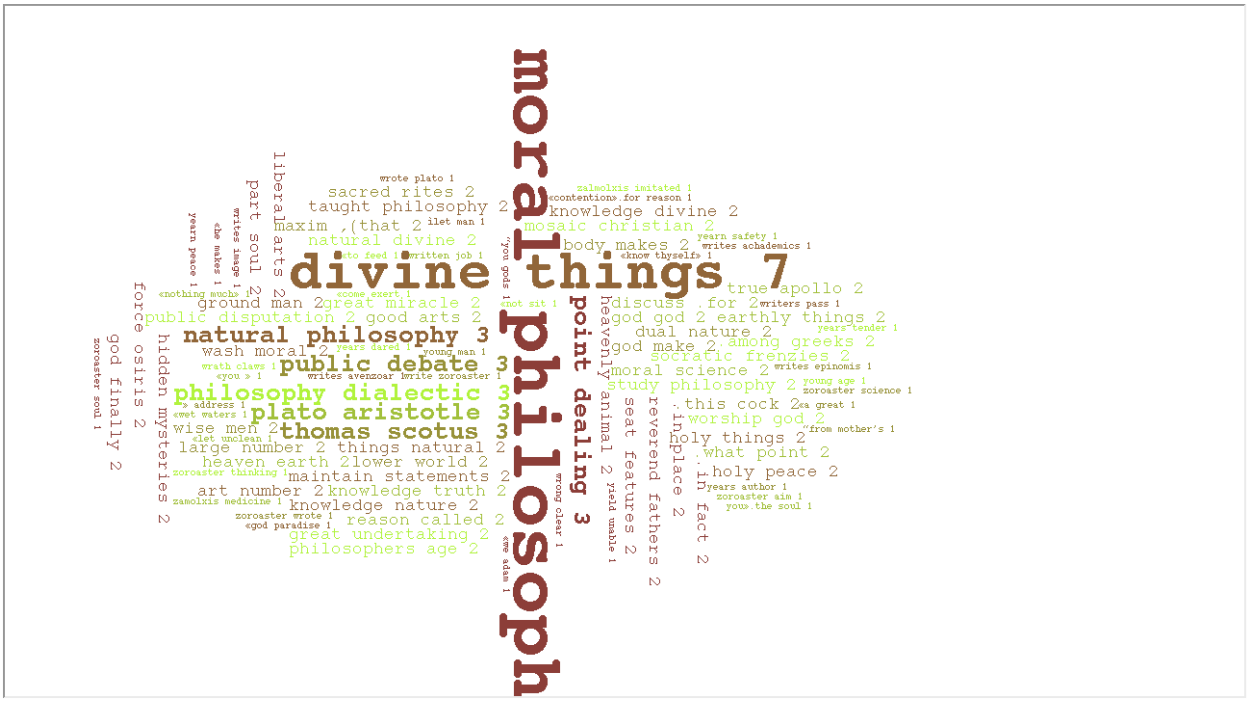


View a cloud of bigrams in an annotated collection of English translations of Italian Renaissance texts from the Brown University Virtual Humanities Lab (http://www.brown.edu/Departments/Italian_Studies/vhl_new/). This example displays bigrams using the modernized/author's first intent filters.

The following flow:

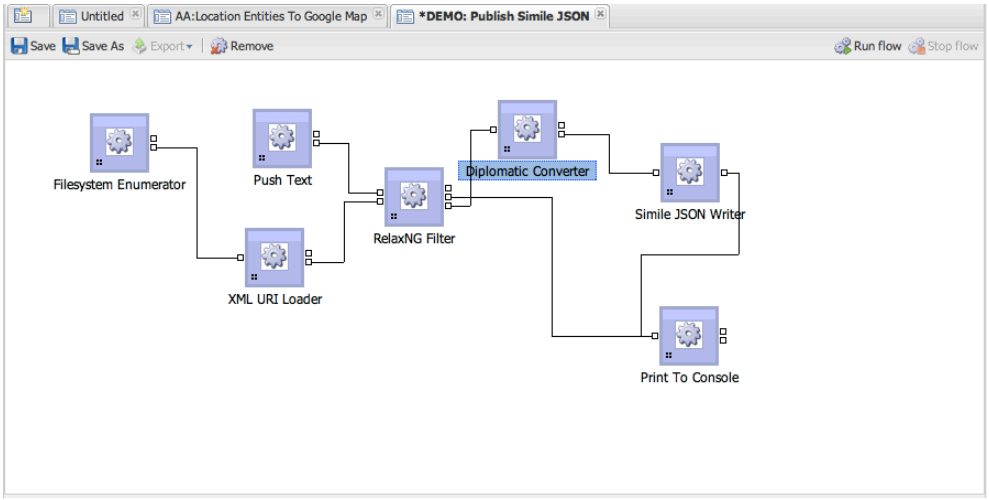


Produces the following result:



View places mentioned in the Decameron in a Simile Exhibit. This example displays places that have been extracted after validation and passing through the diplomatic filters.

The following flow:



Produces the following map: